TP4 : Langages et automates

Objectifs du TP :

- Implémenter quelques opérations sur les automates en Ocaml.
- Faire le lien entre questions sur un langage et algorithmes sur un automate.
- Réviser la manipulation d'enregistrements et du module Array.

Nous avons expliqué en cours que, fondamentalement, résoudre un problème de décision $\mathcal{P}$

> **Entrée :** $x \in \mathcal{I}$ où $\mathcal{I}$ est l'ensemble des instances du problème.
> **Sortie :** Oui si x vérifie une propriété P et non sinon.

revient à déterminer le langage $L_{\mathcal{P}}$ des encodages $\overline{x}$ des éléments x de $\mathcal{I}$ qui satisfont la propriété P. On a ainsi les correspondances suivantes :

Contexte : problème de décision	Contexte : langages
Est-ce que $x \in \mathcal{I}$ est solution du problème $\mathcal{P}$?	Est-ce que $\overline{x}$ appartient à $L_{\mathcal{P}}$?
Y a-t-il une solution au problème $\mathcal{P}$?	Le langage $L_{\mathcal{P}}$ est-il vide ?
Résoudre $\mathcal{P}$ est-il équivalent à résoudre le problème $\mathcal{P}'$?	A-t-on $L_{\mathcal{P}} = L_{\mathcal{P}'}$?

Dans ce TP, on cherche donc à répondre aux trois questions de droite dans le cas où les langages manipulés sont reconnaissables par un automate fini déterministe (avec un seul état initial). On introduit pour représenter de tels objets les types suivants :

```
type etat = int
type lettre = int
type mot = lettre list
```

Un automate déterministe sera représenté par un enregistrement avec les conventions suivantes :

- L'alphabet d'un automate sera toujours $[\![0, m-1]\!]$ avec $m \in \mathbb{N}^\star$.
- Les états d'un automate seront toujours numérotés de 0 à $n-1$ où $n > 0$ est le nombre d'états.
- Les états finaux sont caractérisés par un tableau T de booléens : l'état i est final si et seulement si la case indicée par i dans T contient true.
- La fonction de transition δ d'un automate est caractérisée par la table des transitions delta. Ainsi delta est une matrice de taille $n \times m$ dont la case (q, l) contient l'état $\delta(q, l)$ s'il existe et -1 sinon (c'est-à-dire si lire l dans l'état q provoque un blocage).

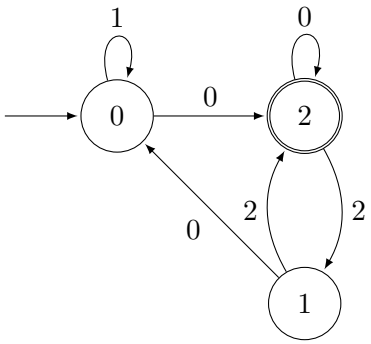
On obtient finalement le type suivant :

```
type automate_det =
  {nb_lettres : int;
   nb_etats : int;
   etat_initial : etat;
   etats_finaux : bool array;
   delta : etat array array};;
```

Exercice 0 *Visualisation d'automates*

Commencez par récupérer le fichier TP4_sujet.ml disponible dans les ressources partagées.

1. Définir une variable a1 de type automate_det représentant l'automate A_1 suivant :



Le fichier partagé fournit deux fonctions permettant de visualiser graphiquement les automates que vous manipulerez et dont voici le fonctionnement. L'appel à

```
graphviz a "test.viz"
```

créé un fichier test.viz donnant une représentation textuelle de l'automate a. Une fois ce fichier créé,

```
genere_pdf "test.viz" "test.pdf"
```

génère un fichier test.pdf qui contient une représentation graphique de l'automate a. Copier-coller le contenu du fichier test.viz dans http://magjac.com/graphviz-visual-editor/ produit le même résultat et peut être une alternative à genere_pdf.

2. Visualiser l'automate A_1 que vous venez de créer puis faire de même pour l'automate A_2 dont la description est fournie dans le sujet. Que constate-t-on sur l'automate A_2 ?

Exercice 1 *Appartenance à un langage*

3. Ecrire une fonction delta_etoile de signature automate_det -> etat -> mot -> etat option telle que delta_etoile a q m renvoie Some q' où q' est l'état obtenu en lisant le mot m depuis l'état q dans a s'il existe et None sinon (c'est-à-dire, s'il y a blocage).

4. En déduire une fonction accepte de signature automate_det -> mot -> bool telle que accepte a m renvoie la réponse à la question "Le mot m est-il reconnu par l'automate a ?".

Constatez que le fait de manipuler un automate déterministe rend le test de l'appartenance d'un mot au langage reconnu par cet automate très simple !

Exercice 2 *Détection de langages vides*

5. Ecrire une fonction etats_accessibles de signature automate_det -> bool array prenant en entrée un automate déterministe a et renvoyant un tableau t contenant true à la case i si et seulement si l'état i est accessible depuis l'état initial dans a.

6. En déduire une fonction langage_non_vide de signature automate_det -> bool renvoyant true si et seulement si le langage reconnu par l'automate en entrée est non vide.

7. Confirmer le bon fonctionnement de cette dernière fonction en la testant sur l'automate A_2 fourni par l'énoncé. Déterminer sa complexité en fonction de grandeurs pertinentes.

8. (bonus) En s'inspirant de la question 5, écrire une fonction est_emonde de signature automate_det -> bool indiquant si l'automate pris en entrée est émondé ou non.

Exercice 3 *Test de l'égalité de deux langages*

Cette partie est à rendre au format papier pour le 14/10.

Dans le début de cette partie, on supposera que les automates considérés sont déterministes, complets, et définis sur le même alphabet. On ne vérifiera pas que c'est bien le cas.

9. Ecrire une fonction complementaire de signature automate_det -> automate_det prenant en entrée un automate déterministe et complet A et renvoyant un automate reconnaissant $L(A)^c$. Le complémentaire est considéré vis-à-vis de l'alphabet Σ constitué des lettres $0, 1, ..., m-1$ où m est le nombre de lettres utilisées dans l'automate A.

10. Ecrire une fonction intersection de signature automate_det -> automate_det -> automate_det prenant en entrée deux automates déterministes et complets A_1 et A_2 reconnaissant respectivement L_1 et L_2 et renvoyant un automate reconnaissant $L_1 \cap L_2$. On pourra commencer par réfléchir au numéro à attribuer à l'état $(q_1, q_2) \in Q_1 \times Q_2$ (où Q_i désigne l'ensemble des états de A_i).

11. Si L et L' sont deux langages sur un même alphabet Σ, caractériser l'inclusion $L \subset L'$ par la vacuité d'un langage L'' construit à partir d'intersections et de complémentarisations sur L et L'. Déterminer L'' et prouver la caractérisation avancée.

12. En déduire une fonction inclusion_langages de signature automate_det -> automate_det -> bool prenant en entrée deux automates déterministes complets et telle que inclusion_langages a1 a2 renvoie true si et seulement si le langage reconnu par a1 est inclus dans celui reconnu par a2.

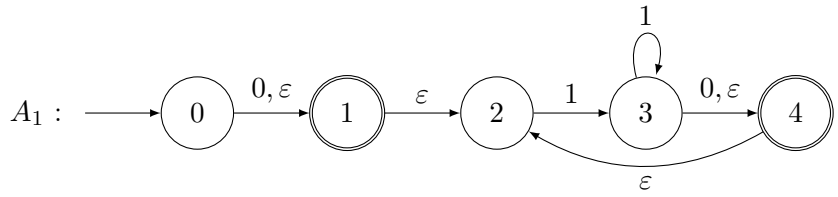
Dans la suite de cette partie, les automates considérés ne sont plus nécessairement complets mais toujours définis sur le même alphabet (sans avoir besoin de le vérifier). Comme vu en cours, complémentariser des automates non complets peut avoir des effets désastreux. Assurons-nous que les fonctions complementaire et par suite inclusion_langages ont des entrées correctes :

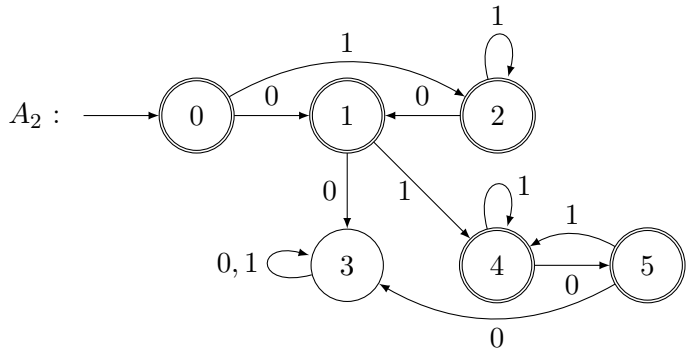
13. Ecrire une fonction est_complet de signature automate_det -> bool indiquant si l'automate pris en entrée est complet.

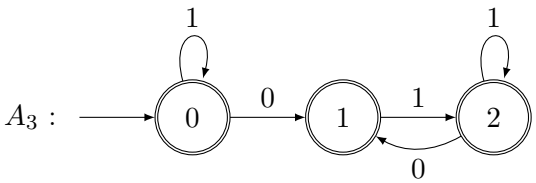
14. Ecrire une fonction complete de signature automate_det -> automate_det prenant en entrée un automate A déterministe complet ou non et renvoyant un automate équivalent à A et complet.

15. Déduire enfin une fonction egalite_langages de signature automate_det -> automate_det -> bool indiquant si les deux automates déterministes (pas forcément complets) en entrée sont équivalents. Déterminer sa complexité en fonction de grandeurs pertinentes.

On considère dans la fin de cette partie les trois automates A_1, A_2 et A_3 suivants :







16. Déterminiser l'automate A_1.
17. En expliquant la démarche, déterminer si A_1 et A_2 sont équivalents. Qu'en est-il de A_1 et A_3 ?
18. Décrire en français les langages reconnus par A_1, A_2 et A_3.